UNIX - SIGNALS AND TRAPS

Signals are software interrupts sent to a program to indicate that an important event has occurred. The events can vary from user requests to illegal memory access errors. Some signals, such as the interrupt signal, indicate that a user has asked the program to do something that is not in the usual flow of control.

The following are some of the more common signals you might encounter and want to use in your programs:

Signal Name	Signal Number	Description
SIGHUP	1	Hang up detected on controlling terminal or death of controlling process
SIGINT	2	Issued if the user sends an interrupt signal (Ctrl + C).
SIGQUIT	3	Issued if the user sends a quit signal (Ctrl + D).
SIGFPE	8	Issued if an illegal mathematical operation is attempted
SIGKILL	9	If a process gets this signal it must quit immediately and will not perform any clean-up operations
SIGALRM	14	Alarm Clock signal (used for timers)
SIGTERM	15	Software termination signal (sent by kill by default).

List of Signals:

There is an easy way to list down all the signals supported by your system. Just issue **kill -l** command and it would display all the supported signals:

```
$ kill -l
 1) SIGHUP        2) SIGINT       3) SIGQUIT       4) SIGILL
 5) SIGTRAP       6) SIGABRT      7) SIGBUS        8) SIGFPE
 9) SIGKILL      10) SIGUSR1     11) SIGSEGV      12) SIGUSR2
13) SIGPIPE      14) SIGALRM     15) SIGTERM      16) SIGSTKFLT
17) SIGCHLD      18) SIGCONT     19) SIGSTOP      20) SIGTSTP
21) SIGTTIN      22) SIGTTOU     23) SIGURG       24) SIGXCPU
25) SIGXFSZ      26) SIGVTALRM   27) SIGPROF      28) SIGWINCH
29) SIGIO        30) SIGPWR      31) SIGSYS       34) SIGRTMIN
35) SIGRTMIN+1   36) SIGRTMIN+2  37) SIGRTMIN+3   38) SIGRTMIN+4
39) SIGRTMIN+5   40) SIGRTMIN+6  41) SIGRTMIN+7   42) SIGRTMIN+8
43) SIGRTMIN+9   44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

The actual list of signals varies between Solaris, HP-UX, and Linux.

Default Actions:

Every signal has a default action associated with it. The default action for a signal is the action that a script or program

performs when it receives a signal.

Some of the possible default actions are:

- Terminate the process.

- Ignore the signal.

- Dump core. This creates a file called core containing the memory image of the process when it received the signal.

- Stop the process.

- Continue a stopped process.

Sending Signals:

There are several methods of delivering signals to a program or script. One of the most common is for a user to type CONTROL-C or the INTERRUPT key while a script is executing.

When you press the *Ctrl+C* key a SIGINT is sent to the script and as per defined default action script terminates.

The other common method for delivering signals is to use the kill command whose syntax is as follows:

```
$ kill -signal pid
```

Here **signal** is either the number or name of the signal to deliver and **pid** is the process ID that the signal should be sent to. For Example:

```
$ kill -1 1001
```

Sends the HUP or hang-up signal to the program that is running with process ID 1001. To send a kill signal to the same process use the folloing command:

```
$ kill -9 1001
```

This would kill the process running with process ID 1001.

Trapping Signals:

When you press the *Ctrl+C* or Break key at your terminal during execution of a shell program, normally that program is immediately terminated, and your command prompt returned. This may not always be desirable. For instance, you may end up leaving a bunch of temporary files that won't get cleaned up.

Trapping these signals is quite easy, and the trap command has the following syntax:

```
$ trap commands signals
```

Here *command* can be any valid Unix command, or even a user-defined function, and signal can be a list of any number of signals you want to trap.

There are three common uses for trap in shell scripts:

1. Clean up temporary files

2. Ignore signals

Cleaning Up Temporary Files:

As an example of the trap command, the following shows how you can remove some files and then exit if someone tries to abort the program from the terminal:

```
$ trap "rm -f $WORKDIR/work1$$ $WORKDIR/dataout$$; exit" 2
```

From the point in the shell program that this trap is executed, the two files *work1$$* and *dataout$$* will be automatically removed if signal number 2 is received by the program.

So if the user interrupts execution of the program after this trap is executed, you can be assured that these two files will be cleaned up. The **exit** command that follows the **rm** is necessary because without it execution would continue in the program at the point that it left off when the signal was received.

Signal number 1 is generated for hangup: Either someone intentionally hangs up the line or the line gets accidentally disconnected.

You can modify the preceding trap to also remove the two specified files in this case by adding signal number 1 to the list of signals:

```
$ trap "rm $WORKDIR/work1$$ $WORKDIR/dataout$$; exit" 1 2
```

Now these files will be removed if the line gets hung up or if the *Ctrl+C* key gets pressed.

The commands specified to trap must be enclosed in quotes if they contain more than one command. Also note that the shell scans the command line at the time that the trap command gets executed and also again when one of the listed signals is received.

So in the preceding example, the value of WORKDIR and $$ will be substituted at the time that the trap command is executed. If you wanted this substitution to occur at the time that either signal 1 or 2 was received you can put the commands inside single quotes:

```
$ trap 'rm $WORKDIR/work1$$ $WORKDIR/dataout$$; exit' 1 2
```

Ignoring Signals:

If the command listed for trap is null, the specified signal will be ignored when received. For example, the command:

```
$ trap '' 2
```

Specifies that the interrupt signal is to be ignored. You might want to ignore certain signals when performing some operation that you don't want interrupted. You can specify multiple signals to be ignored as follows:

```
$ trap '' 1 2 3 15
```

Note that the first argument must be specified for a signal to be ignored and is not equivalent to writing the following, which has a separate meaning of its own:

```
$ trap  2
```

If you ignore a signal, all subshells also ignore that signal. However, if you specify an action to be taken on receipt of a signal, all subshells will still take the default action on receipt of that signal.

Resetting Traps:

After you've changed the default action to be taken on receipt of a signal, you can change it back again with trap if you

simply omit the first argument; so

```
$ trap 1 2
```

resets the action to be taken on receipt of signals 1 or 2 back to the default.